

# L3 informatique - MO1 (2005-2006)

## Modélisation Orientée Objet avec UML

### Introduction

M. Savonnet et M.N. Terrasse  
Département IEM, Université de Bourgogne

30 septembre 2005

---

## 1 Organisation du cours

Cette unité d'enseignement traite de la modélisation des systèmes d'information. Elle est centrée sur la modélisation orientée objet avec UML comme langage de modélisation. L'objectif est de vous présenter UML dans son ensemble, de vous faire percevoir la complémentarité des différents diagrammes qui constituent un modèle UML, de vous faire expérimenter –sur des exemples limités– le choix du diagramme approprié pour décrire un point de vue sur un système d'information.

Certains des diagrammes seront vus en détail (diagramme Use Case, diagramme de classe et diagramme StateChart) ; les autres diagrammes seront brièvement présentés. Le planning des cours, TD et TP est donné en Figure 1.

## 2 Introduction à l'Orienté Objet

### 2.1 Point de vue historique

Si l'on remonte très loin dans l'histoire de la programmation, les premiers logiciels de taille importante ont été développés selon la règle du *tout débogage*. Le bilan a été lourd : retards de plusieurs années, coûts effectifs sans rapport avec les devis, manque de fiabilité et mauvaises performances, difficultés de maintenance. Ceci a conduit à la crise du logiciel de la fin des années 60 durant laquelle on se rend compte que le développement du logiciel coûte beaucoup plus cher que le matériel. Et on ne parle pas encore des coûts de maintenance ! Le génie logiciel travaille à la définition de méthodes adaptées aux logiciels de taille critique : travail en équipes, méthodes d'ingénierie, mise en place du système avec prise en main par les utilisateurs. Pendant toutes les années 70, la notion de performance prend de l'importance.

Des modèles du cycle de vie du logiciel se multiplient, assortis de règles de développement appropriées. Les qualités essentielles des logiciels sont alors : *fiabilité, évolutivité, fidélité aux demandes des utilisateurs et possibilité de réutilisation du code*.

Ainsi qu'illustré en Figure 2, la programmation objet est issue de la convergence de trois courants de l'informatique : le génie logiciel, l'intelligence artificielle, les bases de données.

**La branche génie logiciel** vise à établir des règles de développement de grosses applications. Elle a produit plusieurs des concepts qui ont fait référence dans les années 70. Ce sont, entre autres, la *modularité* et la *programmation structurée*. La notion de module est devenue un classique dans tous les langages de programmation. Elle permet de construire un programme à partir de plusieurs composants. Dans chacun de ces composants, on veut pouvoir distinguer la partie descriptive du composant (ce qu'il fait) de la partie implémentation (comment il le fait). La technique de compilation séparée a permis à la programmation structurée de prendre forme et d'être efficace dans la mesure où une application peut être construite comme un assemblage de boîtes à outils (appelées modules). La programmation structurée est issue de la méthode d'analyse de problème par

semaine 26-30 sept	CM1	Introduction <i>Projet : Distribution de l'énoncé</i>
semaine 3-7 oct	CM2	Le diagramme Use Case
semaine 10-14 oct	CM3 TD1	Le diagramme de classe Diagramme Use Case
semaine 17-21 oct	CM4  TD2	Automates temporisés, Diagramme statechart & diagramme de collaboration Diagramme Use Case & diagramme de classe <i>Projet : partie étude bibliographique à rendre</i>
semaine 24-28 oct	CM5 TD3	Autres diagrammes Diagramme de classe
semaine 31 oct - 4 nov	TD4 TP1	Automates temporisés & diagramme statechart Diagramme de classe et Objecteering
semaine 7-11 nov	TD5 TP1	Automates temporisés & diagramme statechart Diagramme de classe & Objecteering (rattrapage)
semaine 14-18 nov	TD6	Autres diagrammes : timing, package, composant, déploiement, interaction
semaine 21-25 nov	TD7	Exemples de modélisation : e-learning, librairie en ligne, PPE, Apogée, ... <i>Validation des diagrammes du projet</i>
semaine 28 nov - 2 déc	TD8 TP2	Choix des diagrammes Passage d'UML à SQL <i>Projet : rapport final de modélisation à rendre</i>

FIG. 1 – Planning des cours, TD et TP pour l'année 2005-2006

raffinements successifs : chaque problème est décomposé en sous-problèmes qui seront eux-mêmes décomposés. Le processus de décomposition s'arrête lorsque les problèmes obtenus sont d'une complexité assez faible pour être traités directement. La convergence des deux concepts (programmation structurée et modularité) conduit à associer à chaque problème non décomposé un module de traitement.

Les langages phares de cette branche sont les langages Algol (plusieurs versions de 1960 à 1968, qui ont largement étudié la notion de sous-programme et les modes de passages de paramètres) ainsi que le langage Pascal (1971). La théorisation est due en majeure partie à Wirth avec son ouvrage "Algorithms + Data Structure = programs", publié en 1976.

Au niveau recherche, le début des années 80 a vu l'émergence de la notion de *type abstrait de données* avec, par exemple, la notion de "boîte noire" proposée par Goguen. Cette boîte noire permet de spécifier les opérations et leurs relations par des axiomes, sans souci d'implémentation.

**La branche intelligence artificielle** a pour but de modéliser le raisonnement humain ce qui la conduit à chercher des méthodes d'utilisation de masses importantes d'information non structurée. Les méthodes proposées cherchent à regrouper différentes informations autour de concepts communs. Les langages phares de cette branche sont le langage Simula (1966) qui met en place les notions de classes et d'objets ainsi que les langages de la famille Smalltalk. La théorisation repose sur les travaux de Liskov et Guttag (1970).

A partir de 1973, les *langages acteurs* introduisent la notion de connaissance distribuée et de communication entre unités par envoi de messages. Enfin les *réseaux sémantiques* mettent en place la plupart des concepts utilisés

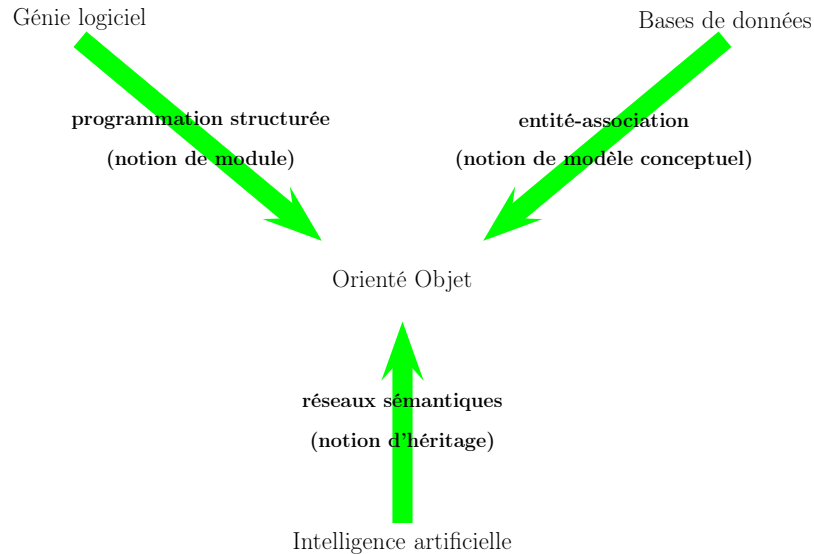


FIG. 2 – L’Orienté Objet au croisement de trois courants

à l’heure actuelle (dont la notion d’héritage, Winston, 1975) pour structurer des informations en généralisant les concepts communs. L’abstraction est présentée comme un mécanisme permettant de se “concentrer sur l’essentiel en oubliant les détails”.

**La branche base de données** apporte de nombreux travaux sur la modélisation avec en particulier les modèles sémantiques, dont le *modèle entité-association* de Chen (1976). Les réflexions issues des manifestes pour les BD de 3<sup>ième</sup> génération (manifeste initial par Atkinson & al. en 1989, réponse dans ACM de septembre 1990) mettent en avant les notions de généralisation-spécialisation.

La branche BD ajoute la persistance des données et la normalisation des modélisations. Elle introduit la notion d’*identifiant d’objet* pour limiter la redondance (travaux de Kuper et Vardi en 1985). Plusieurs SGBD OO apparaissent à partir du milieu des années 80 : O2, Orion, Objectstore, Ontos, Versant, Caché. Ces SGBD ont disparu depuis au profit de systèmes hybrides (objet-relationnel).

## 2.2 Les concepts fondamentaux

L’idée fondatrice de l’Orienté Objet et de ses évolutions est l’abstraction. La Figure 3 présente un exemple d’abstraction dans la façon de voir les informations en informatique : de la zone mémoire à la notion de type abstrait de données (d’autres exemples sont donnés dans l’exercice ci-dessous). L’abstraction prend en modélisation plusieurs formes (généralisation, points de vue, etc) que nous allons détailler.

### Exemple 1- Abstraction

De nombreux exemples d’abstraction existent en informatique. Essayez de retrouver sur quelle caractéristique se fonde l’abstraction dans les exemples suivants :

- i) Le diagramme de classe UML et le modèle relationnel.
- ii) Un câble réseau et un Vlan.
- iii) Une arborescence Unix et un disque virtuel.
- iv) Les adresses physique (ethernet), logique (internet) et noms de domaines en réseau.
- v) Adressage mémoire relatif et la mémoire virtuelle à base de pagination.

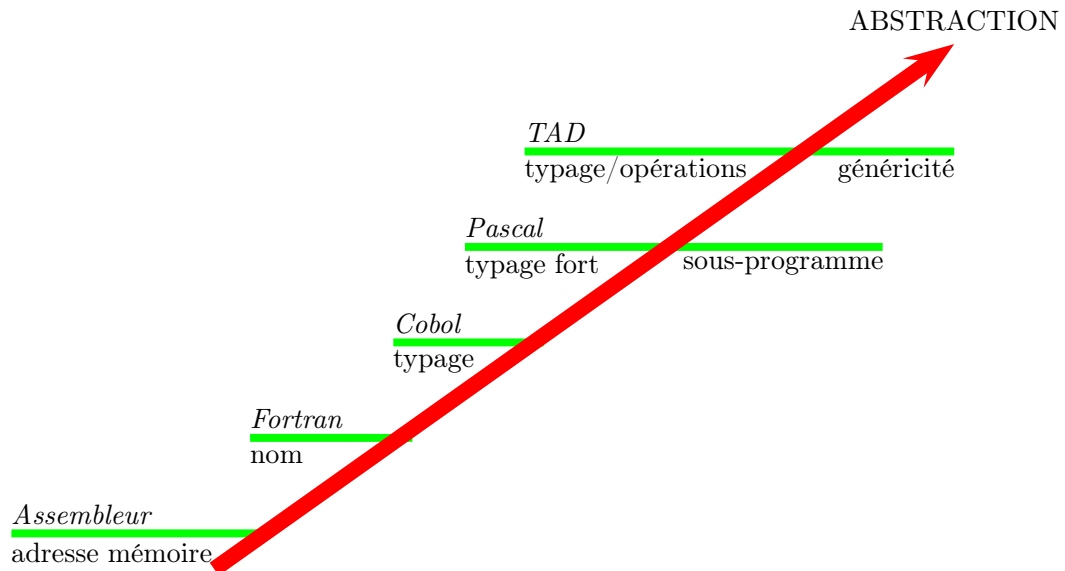


FIG. 3 – Mécanisme d'abstraction

**La notion de généralisation** La généralisation est la forme la plus connue de l'abstraction. Ce terme est celui utilisé en base de données (généralisation-spécialisation) mais la notion d'héritage utilisée en génie logiciel est très proche de la généralisation. La généralisation-spécialisation permet de rattacher plusieurs classes d'objets présentant des similarités à une même classe qui décrit uniquement leurs propriétés communes. Cette vision de l'abstraction n'est pas propre à l'informatique : elle a été largement mise en œuvre par les grandes classifications taxonomiques (plantes, animaux) ou philosophiques (les catégories des philosophes grecs).

**La notion de points de vue** En réaction à la complexité des systèmes à traiter et à la croissance des contraintes à prendre en compte, l'Orienté Objet a instauré un parti-pris de fractionnement qui n'est plus basé sur une décomposition a priori du problème (comme en programmation structurée) mais sur l'utilisation de plusieurs points de vue ou aspects orthogonaux (i.e., si possible indépendants les uns des autres) mais traités de façon cohérente. L'image la plus simple que l'on peut utiliser pour décrire le mécanisme des points de vue est celui d'un puzzle : chaque pièce du puzzle correspond à un point de vue indépendant mais pour pouvoir reconstituer le puzzle, il faut disposer de toutes les pièces et il faut que les pièces s'emboîtent correctement.

La première version de ce fractionnement a été de considérer de façon séparée les aspects statiques et dynamiques des objets. Dans la description d'une classe les attributs peuvent être statiques (les champs) ou dynamiques (les opérations), c'est l'encapsulation qui garantit la cohérence des deux. Les langages de modélisation sont allés beaucoup plus loin que les *aspects* avec, par exemple pour UML, l'utilisation de *points de vue* tels que la vue externe du système, la vue statique, la vue dynamique, la vue des composants, ... qui sont eux mêmes décomposés en *diagrammes*.

### 3 Introduction à UML

De nombreuses méthodes Orientée Objet (OO) ont vu le jour dans la première moitié des années 90. Chacune d'entre elles a sa vision propre de l'OO et du processus de modélisation. À partir de 1994, l'OMG (Object Group Management, [www.omg.org](http://www.omg.org)) et les organisateurs de la conférence OOPSLA, réclament un standard OO. En 1995, pendant la conférence OOPSLA, Booch, Rumbaugh et Jacobson présentent une première proposition UML (Unified Modeling Language [13, 16]).

Aspects O.O.	Vues des systèmes multi-vues	Diagrammes UML2.0
Statique	Objet	Diagramme d'objet
	Physique	Diagrammes de composants, de déploiement de package
	Structurelle	Diagrammes de classe, des structures composites
Dynamique	Externe	Use Case
	Comportementale	Diagrammes d'activité, de séquence, de description des interactions, de timing, de communication, statechart

FIG. 4 – UML2.0 : aspects, vues et diagrammes

Le terme de “langage” est d’ailleurs très réducteur dans la mesure où UML est composé de trois parties : une notation (le langage proprement dit) mais aussi un métamodèle (qui décrit la notation) et une méthode de développement [10] proposée en 1999. L’OMG vient de publier la version finale de la spécification d’UML2.0<sup>1</sup>.

Le langage UML2 est articulé autour de 13 diagrammes. Ces diagrammes constituent des vues convergentes du système d’information. Les vues relèvent des aspects statiques ou dynamiques (voir Figure 4). Ces diagrammes sont : le diagramme de classe, des structures composites, de composants, de déploiement, d’objet, de package, d’activité, de séquence, de description des interactions, de timing, de communication, ainsi que les diagrammes stateChart et Use Case.

**La vue externe** (diagramme Use Case) décrit le système sous forme d’un ensemble de fonctionnalités (appelées *use cases*) qui sont offertes à différents types d’utilisateur (appelés *acteurs*).

**La vue structurelle** (diagramme de classe et diagramme des structures composites) est le noyau dur de la description statique du système. Le diagramme de classe rend compte des entités en jeu ainsi que des relations “sémantiquement fortes” entre ces entités. Le diagramme des structures composites permet de mettre en valeur les interfaces entre classes.

#### Exemple 2- Relations entre entités

Philippe Desfray [3] avait proposé de considérer trois niveaux de relations entre classes. Les relations les plus fortes correspondent à un lien sémantique entre objets. Les relations intermédiaires correspondent au cas d’un objet d’une classe utilisé comme paramètre d’une opération d’une autre classe. Les relations faibles correspondent au cas d’un objet d’une classe utilisé comme variable locale dans le corps d’une méthode d’une autre classe.

Soient les classes ENSEIGNANT, COURS, SALLE, FILIÈRE<sup>2</sup>. Utilisez cette définition pour décider de l’importance des relations induites par les informations suivantes :

vi) Un enseignant est responsable de cours.

vii) Une filière est constituée de cours.

viii) Un cours a lieu dans une salle.

ix) Sur le planning d’une salle figure le nom du responsable de cours.

x) On doit pouvoir construire la liste des enseignants qui ont besoin du code d’accès à une salle.

xi) Dans le cadre de la LOLF<sup>3</sup>, il faut pouvoir indiquer pour chaque salle la répartition de son utilisation entre les niveaux licence et master.

<sup>1</sup>UML Superstructure Specification, v2.0, juillet 2005, <http://www.omg.org/cgi-bin/doc?formal/05-07-04>.

<sup>2</sup>Par exemple L3 informatique, L2 math-info, M1 multimédia, M2 BD-IA sont des filières.

<sup>3</sup>LOLF : Loi Organique relative à la Loi de Finances.

**La vue comportementale** (diagrammes de séquence, d'activité, de communication, de description des interactions, de timing et statechart) décrit la dynamique du système. Ces diagrammes établissent (soit pour une classe isolée, soit pour plusieurs classes en interaction) les protocoles à travers lesquels les objets communiquent et interagissent.

**La vue physique** (diagrammes de package, des composants, de déploiement) décrit le système comme un ensemble de composants (packages de données, code, documentation, ...) qui sont positionnés sur des machines reliées par les liens physiques d'un réseau. La description des types de liens entre sites, combinée avec le modèle des interactions entre composants, est une base pour les spécialistes sécurité ([11] pour plus de détails).

**La vue objet** (Object Diagram) présente le système en terme d'objets reliés. C'est à la fois un "instantané" du système et une instanciation du diagramme de classe.

## 4 Bibliographie

Voici quelques références de livres qui me semblent intéressants. Vous pouvez aussi chercher sur internet.

Une excellente présentation synthétique : [7] en anglais et sa version française [8].

Les ouvrages de présentation détaillés : [5, 14].

Les descriptions officielles : [1, 10, 13, 15, 17]

Les livres sur UML2.0 : [6, 4]

UML et langages de programmation : [12].

Des exemples de développement de modèles : [9].

Une bonne référence sur l'orienté objet : [2].

## Références

- [1] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language - User Guide*. Addison-Wesley, 1998. ISBN 0-201-57168-4.
- [2] Chabane Oussalah. *Ingénierie objet : concepts et techniques*. InterEditions, 1997.
- [3] Philippe Desfray. *Modélisation par objets - la fin de la programmation*. InterEditions, 1997.
- [4] L. Doldi. *UML 2 Illustrated - Developing Real-Time and Communication Systems*. TMSO, 2004. ISBN 2-9516600-1-4.
- [5] P. Evitts. *A UML Pattern Language*. Macmillan Technical Publishing, USA, 2000. ISBN 1-57870-118-X.
- [6] M. Fowler. *UML 2.0*. Campus Press, 2004. ISBN.
- [7] Martin Fowler and Kendall Scott. *UML Distilled - Applying the Standard Object Modeling Language*. Addison-Wesley, 1998. ISBN 0-201-65783-X.
- [8] Martin Fowler and Kendall Scott. *UML*. CampusPress, Collection "Le tout en poche", 2001. ISBN 2-7440-1090-1, traduction par M.C. Baland et L. Carité, titre original "UML Distilled, Second Edition".
- [9] Hassan Gomaa. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley, 2000. ISBN 0-201-65793-7.
- [10] Ivar Jacobson, Grady Booch, and James Rumbaugh. *Le Processus unifié de développement de logiciel*. Eyrolles, 2000. ISBN 0-201-57169-2, (Titre original : The Unified Software Development Process, Addison-Wesley).
- [11] Jan Jürjens. *Towards Development of Secure Systems Using UML*, pages 187–200. LNCS 2029, 2001.
- [12] William M. Tepfenhart Richard Leei. *UML et C++*. Prentice Hall, 1998.
- [13] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language - Reference Manual*. Addison-Wesley, 1998. ISBN 0-201-30998-X.
- [14] S. Si Alhir. *UML in a Nutshell*. O'Reilly, 1998. ISBN 1-56592-448-7.
- [15] Jos Warmer and Annekke Kleppe. *The Object Constraint Language - Precise Modeling with UML*. Addison-Wesley, 1999. ISBN 0-201-37940-6.

- [16] *OMG Unified Modeling Language Semantics*, June 1999. Version 1.3, Available at URL <http://www.omg.org>.
- [17] *OMG Unified Modeling Language Specification*, March 2000. Version 1.3, Available at URL <http://www.omg.org>.